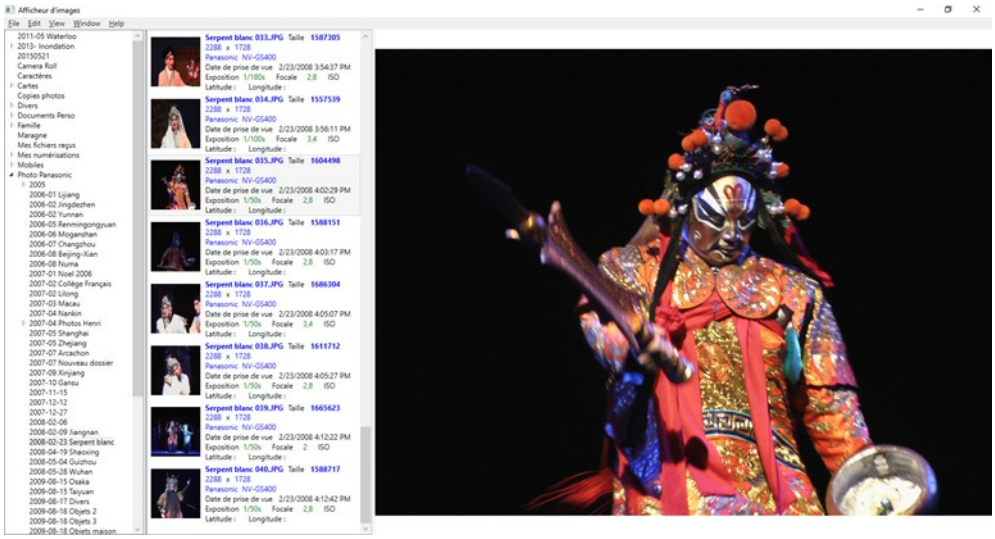
Afficheur d’images WPF 2022

Objectif

Il s’agit de créer une application permettant de parcourir le répertoire « Mes Images », d’afficher la liste des images qu’il contient, d’afficher en plus grand format une image sélectionnée et de créer un diaporama.



Pour pouvoir tester notre application, un jeu d’essai sera évidemment nécessaire, qui devra être situé sous ce répertoire et comporter au moins deux répertoires d’images. Si les images que l’on peut télécharger sur Internet sont évidemment utilisables, de « vraies » photos issues d’un appareil photo (ou même d’un téléphone) sont préférables car porteuses de beaucoup plus d’informations. Une partie de l’exercice exploitera le fait que ces photos sont au format *JPEG*. Si nécessaire, on trouvera sur le moodle du cours un jeu d’essai adapté, mais vous pouvez naturellement utiliser des photos personnelles.
Nous allons développer cette application en nous appuyant sur la technologie *Windows Presentation Foundation*, en utilisant un modèle de conception Modèle/Vue/Vue-Modèle, en utilisant largement la technique du data binding, et en écrivant le minimum de code possible.

Ne pas hésiter à consulter régulièrement : https://www.wpf-tutorial.com/

I. **Premiers pas**

1) Création du projet.
Ouvrir VS 2022, créer un projet "Visual C#/Application WPF", donner un titre comme "AfficheurWPF", sélectionner : l'emplacement ("Documents/..."), la version la plus à jour de .Net (6.0), valider.

2) Personnalisation.
Dans le fichier "MainWindow.xaml", cliquer sur la fenêtre (pas la grille) et changer sa propriété (en bas à droite) "Title" à quelque chose comme "Afficheur d'images". On constate que le titre change a) là où il est tapé, b) dans la fenêtre "graphique" du concepteur, c) dans le code XAML, et d) dans la fenêtre de l'application qu'on peut tester aver "Démarrer".

Puis, cliquer sur la grille (au centre de la fenêtre). Dans la catégorie "Disposition", cliquer sur le bouton "..." à côté de "RowDefinitions".

On obtient une fenêtre permettant de définir les lignes de la grille. Ajouter un nouvel élément dont on changera la hauteur (premier contrôle) à "20" en "Pixel". Ajouter un second élément qu'on laisse à "1" en "Star". Valider. Si tout va bien, on devrait avoir le code suivant en XAML :

```
<Grid.RowDefinitions>
    <RowDefinition Height="20"/>
    <RowDefinition/>
</Grid.RowDefinitions>
```

4) Edition directe du XAML

Copier-coller ceci après "RowDefinitions" :

```
<Grid Grid.Row="1">
    <!-- #region Structure -->
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="150"></ColumnDefinition>
        <ColumnDefinition Width="5"></ColumnDefinition>
        <ColumnDefinition Width="100"></ColumnDefinition>
        <ColumnDefinition Width="5"></ColumnDefinition>
        <ColumnDefinition Width="*"></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <!-- #endregion -->
    <GridSplitter Grid.Column="1" ResizeBehavior="PreviousAndNext" Width="3" />
    <GridSplitter Grid.Column="3" ResizeBehavior="PreviousAndNext" Width="3"/>
</Grid>
```

Explications :

Ceci insère une seconde grille dans la première (définie par ses deux lignes). La deuxième grille contient cinq colonnes. Cela revient à poser un tableau dans une case d'un premier tableau. La première ligne <Grid Grid.Row="1"> signifie qu'on crée un tableau qui s'insèrera dans la deuxième ligne du premier tableau (les lignes étant 0 et 1). "Row" est une "propriété attachée" : on dit que l'OBJET, instance de Grid, que l'on crée (premier "Grid") se situe dans un tableau à la ligne 1 (la deuxième) en appelant une propriété associée à la CLASSE Grid (deuxième "Grid"), qui n'est valable que si cet objet est placé lui-même dans un objet de type Grid (la balise englobante).
Ensuite, on définit ses cinq colonnes : 0 = contenu, 1 = séparateur, 2 = contenu, 3 = séparateur, et 5 = contenu. Les valeurs signifient que les largeurs initiales sont fixes sauf pour la dernière colonne, qui occupera « le reste de la place disponible ».
Ensuite, on ajoute des séparateurs (GridSplitter), donc en colonnes 1 et 3, dont la propriété "ResizeBehavior" permet à l'utilisateur de redimensionner librement les colonnes autour (donc 0 et 2 pour le premier, 2 et 4 pour le second). Il n'y a qu'une ligne dans ce tableau. On peut imbriquer autant de tableaux que l'on veut…
Les commentaires sont des commentaires XML standard <!-- Comment... -->, mais on peut utiliser les macros VS comme #region à l'intérieur de ces commentaires.

5) Ajout d'un contôle standard "ListBox"

Depuis la "boîte à outils", glissez-déposez une "ListBox" sur la colonne 2 (troisième colonne en tout, deuxième colonne de contenu, 100px de large). Il est possible que vous deviez rectifier da position/taille/marge/... à l'aide des propriétés ou du concepteur. Changez aussi sa propriété "Nom" (à part, en haut dans la fenêtre de propriétés) à quelque chose comme "listBox". Si vous n'y arrivez pas, vous pouvez aussi utiliser le XAML. Le résultat devrait être :

```
<ListBox x:Name="listBox" Grid.Column="2" Margin="0"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>
```

Ceci placé dans la seconde balise "Grid" (après les GridSplitter ; l'attribut "Margin" est optionnel). La propriété "Nom" (x:Name dans le XAML) est primordiale : elle permet de nommer l'instance de ListBox et d'y faire référence dans du code C#.

III. **Les données de l'afficheur d'images**

1) Que va-t-on afficher ?
On cherche à donner à voir les fichiers d'un dossier choisi par l'utilisateur (par défaut, "Mes Images"). Parmi ces fichiers, ceux qui sont des images seront conservés. Pour ce faire, on va créer une "source de données" sous la forme d'une liste de fichiers dans une classe prévue à cet effet.

2) Classe "FileSource"
Créer une nouvelle classe. Voici son code temporaire :

```
public class FileSource
{
    public List<FileInfo> Files { get; set; }
    public FileSource (string path)
    {
        Files = new List<FileInfo>();
        Directory.GetFiles(path)?.ToList().FindAll(f =>
f.ToLower().EndsWith("jpg")).ForEach(f=>Files.Add(new FileInfo(f)));
    }
}
```

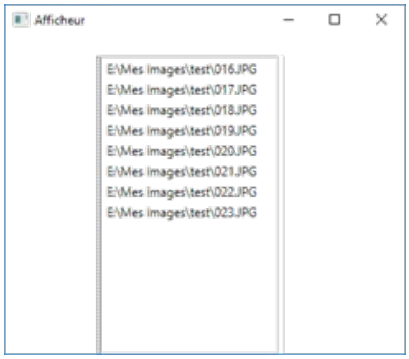
Elle aura besoin d'un "using System.IO;".

Explications :
Cette classe se contente de gérer une liste de fichiers d'un répertoire donné par "path". A la construction de la classe, l'ensemble des fichiers terminant par "jpg" (en majuscule ou minuscule grâce à l'appel à ToLower) est ajouté à cette lise. Noter l'appel à ToList (Directory.GetFiles renvoyant un "simple" tableau), le ?. (qui permet de ne pas tenter d'agir sur un élément null, renvoyé par cette méthode si jamais le chemin est incorrect, inaccessible, ou que le dossier est vide).

3) Liaison
Revenez à la ListBox de la fenêtre principale. On lui ajoute un attribut ItemsSource="{Binding Files} (dans le XAML, c'est plus simple).
Le principe est le suivant : une ListBox affiche un ensemble de ligne qui sont des Items. On peut éditer cette propriété manuellement, en la manipulant comme une liste, en C#, ou signifier qu'elle est déduite automatiquement d'une autre liste. Le "Data Binding" permet de réaliser cette manipulation. La ligne en question fait appel à une propriété "Files" qui est définie "au bon endroit". Par exemple, les fenêtres WPF disposent (entres autres) d'une propriété "DataContext" qui permet de définir une source de données (arbitrairement, donc de type "object"). Pour la positionner correctement, il faut la positionner. Dans le code du constructeur, dans MainWindow.xaml.cs, ajoutez la ligne :

```
        DataContext = new
FileSource(Environment.GetFolderPath(Environment.SpecialFolder.MyPictures));
```

4) Test
Posez quelques fichiers .jpg dans votre dossier "Mes Images" et exécutez l'application. Vous pouvez agrandir la fenêtre contenant le nom des fichiers. Vous pouvez constater que l'affichage actuel donne le chemin absolu de chaque image et rien d'autre : en l'absence d'autres directives de mise en forme, la ListBox applique simplement la méthode ToString aux objets à afficher, qui sont des FileInfo contenant donc ce chemin d'accès.



IV. Mise en forme de la liste et des images

1) Introduction

Pour les contrôles affichant plusieurs objets (liste Items), on peut spécifier un Template donnant une mise en forme par un ou plusieurs éléments graphiques. Ici, nous allons poser un ensemble permettant d'afficher une ligne de la ListBox sous la forme d'une image et de plusieurs blocs de texte. le Template défini contient les éléments suivants :

- Image : une image (vignette) prévisualisant le contenu de la photo.
- TextBlock : du texte mis en forme (on peut lui donner des propriétés comme police, couleur, etc.) permettant d'afficher quelques
informations sur l'image.
- StackPanel : un élément permettant de regrouper d'autres éléments. Son orientation détermine l'agencement (automatique) des éléments
à l'intérieur. On peut imbriquer des StackPanel.

Donc : chaque ligne contiendra un StackPanel horizontal permettant d'afficher l'image à gauche et les données à droite. Les données seront composées d'un StackPanel vertical (pour avoir plusieurs lignes), qui contiendra des StackPanel horizontaux (un par ligne), dont chacun contiendra des TextBlock (affichant les données).

2) Template en XAML

Remplacez votre ListBox dans MainWindow.xaml par la suivante :

```
            <ListBox x:Name="listBox" Grid.Column="2" Margin="0"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch" ItemsSource="{Binding
Files}">
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <StackPanel Orientation="Horizontal">
                            <Image Width="80" Height="60" HorizontalAlignment="Center">
                                <Image.Source>
                                    <BitmapImage DecodePixelWidth="80"
DecodePixelHeight="60" UriSource="{Binding Path=FullName}" />
                                </Image.Source>
                            </Image>
                            <StackPanel Orientation="Vertical">
                                <StackPanel Orientation="Horizontal">
                                    <TextBlock Text="{Binding Path=Name}"/>
                                    <TextBlock Text="Taille"/>
                                    <TextBlock Text="{Binding Path=Length}"/>
                                </StackPanel>
                            </StackPanel>
                        </StackPanel>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
```

3) Explications détaillées

- La ListBox a maintenant un CONTENU (les instructions de mise en forme), on remplace donc la balise <ListBox.../> par <ListBox>...</ListBox>.
- On spécifie la propriété attachée ItemTemplate de la ListBox : c'est ce qui permettra de mettre en forme chaque Item.
- Cet élément est un "Template", ici un DataTemplate.
- Il renvoie les éléments à afficher (un StackPanel contenant d'autres éléments).
- L'image est définie par des attributs classiques (taille) et sa propriété rattachée "Source". Ce dernier élément est une BitmapImage définie
par une URI (emplacement sur le disque ou adresse web).

- L'URI de la source de l'image et le Text des TextBlock sont "liés" à des propriétés. L'idiome "{Binding Path=XXX}" indique que la propriété XXX de l'item à afficher va être utilisée : ici, il s'agit du FileInfo de chacun des fichiers finissant par .jpg.

4) Test

Essayez. On constate un problème de mise en forme qui peut se régler assez facilement (tout est "collé", il y a une propriété "Margin" à adapter).

5) Données EXIF

Récupérez la bibliothèque EXIF (Exif.zip) sur Moodle et ajoutez une référence à exif.dll (préalablement extraite) dans votre projet. Dans la classe "FileSource", ajoutez une directive "using Exif;" et changez le code pour faire référence au type "JPGFileData" comme ceci :

```
    public class FileSource
    {
        public List<JPGFileData> Files { get; set; }
        public FileSource (string path)
        {
            Files = new List<JPGFileData>();
            Directory.GetFiles(path)?.ToList().FindAll(f =>
f.ToLower().EndsWith("jpg")).ForEach(f=>Files.Add(new JPGFileData(f)));
        }
    }
```

Le reste du code devrait continuer à fonctionner. Chaque item de la ListBox dépend maintenant d'un objet de type JPGFileData construit par la bibliothèque Exif (M. Bauderon, 2017). On peut donc utiliser les mêmes propriétés que précédement (qui existent toujours), mais aussi les données Exif de photographies, si elles sont présentes (attention, la plupart des champs sont ignorés et optionnels, surtout si vos images ne sont pas des photographies).

6) Mise en forme complète

Le "Data Binding" est (un peu trop) détaillé ici :
https://docs.microsoft.com/fr-fr/dotnet/framework/wpf/data/data-binding-wpf

Voici un exemple de mise en forme plus complète (ce XAML suffit). Les explications suivront.

```
     <ListBox x:Name="listBox" Grid.Column="2" Margin="0" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" ItemsSource="{Binding Files}">
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <StackPanel Orientation="Horizontal">
                            <Image Width="80" Height="60" HorizontalAlignment="Center">
                                <Image.Source>
                                    <BitmapImage DecodePixelWidth="80"
DecodePixelHeight="60" UriSource="{Binding Path=FullName}" />
                                </Image.Source>
                            </Image>
                            <StackPanel Orientation="Vertical" Margin="2,2,5,2">
                                <StackPanel Orientation="Horizontal" Margin="0,2,2,2">
                                    <TextBlock Text="{Binding Path=Name}"
Margin="0,0,2,0" Foreground="Blue"/>
                                    <TextBlock Text="Taille : " />
                                    <TextBlock Text="{Binding Path=Length}"
Margin="0,0,2,0" FontWeight="Bold"/>
```

```
                                </StackPanel>
                                <StackPanel Orientation="Horizontal" Margin="0,2,2,2">
                                        <TextBlock Text="Pris le : " Margin="0,0,2,0"/>
                                        <TextBlock Text="{Binding Path=DateTaken}"
Margin="0,0,2,0" Background="GhostWhite"/>
                                </StackPanel>
                                <StackPanel x:Name="positionPanel"
Orientation="Horizontal" Margin="0,2,2,2">
                                        <TextBlock Text="{Binding Path=Latitude}"
Margin="0,0,2,0" FontStyle="Italic"/>
                                        <TextBlock Text="x" Margin="0,0,2,0"
FontStyle="Italic"/>
                                        <TextBlock Text="{Binding Path=Longitude}"
Margin="0,0,2,0" FontStyle="Italic"/>
                                </StackPanel>
                            </StackPanel>
                        </StackPanel>
                        <DataTemplate.Triggers>
                            <DataTrigger Binding="{Binding Path=Latitude}">
                                <DataTrigger.Value>
                                    <x:Null/>
                                </DataTrigger.Value>
                                <Setter TargetName="positionPanel"
Property="Visibility" Value="Hidden"/>
                            </DataTrigger>
                        </DataTemplate.Triggers>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
```

7) Plus d'explications

On met en forme différentes propriétés avec la bibliothèque Exif. Un Binding utilise généralement deux éléments : Source (pour désigner l'objet visé : ici, la propriété ItemsSource s'en occupe) et Path (désignant le chemin de la propriété visée). Ici, toutes les propriétés (FullName, Name, Length, DateTaken, Latitude, Longitude) sont des propriétés du type JPGFileData (visualisable dans la bibliothèque importée) et sont des "string". C'est assez facile d'utilisation. Il y a quelques fantaisies de mise en forme : Foreground (couleur de texte), Background (couleur de fond), FontStyle (italique…) et FontWeight (gras...). Le positionnement relatif des blocs de texte peut être fait avec Margin (quatre directions, inutile de mettre une marge à gauche si l'élément de gauche a déjà une marge à droite) et/ou simplement avec des espaces dans le texte.

Plus intéressant est le caractère optionnel des propriétés. Dans une photographie par exemple, les coordonnées GPS peuvent ne pas être renseignées. Si c'est le cas, la bibliothèque Exif donne à la propriété concernée la valeur "null" ; et la mise en forme se fait mal (dans le code précédent, on peut simplement avoir "x" sans les coordonnées GPS autour). Pour éviter ce mauvais rendu, il y a beaucoup de solutions. L'une d'entre elles consiste à créer, dans la classe définissant la source des données, des propriétés "Visibility" correspondant au fait de devoir afficher ou non certains éléments ; comme il s'agit d'une classe importée (fermée à la modification, principe de la programmation objet), ce n'est pas (directement) possible. La solution la plus simple consiste à utiliser un déclencheur ("Trigger").

Dans l'exemple d'au-dessus, on souhaite ne pas afficher la ligne concernant les coordonnées GPS si celles-ci ne sont pas définies. On commence par NOMMER cette ligne (attribut x:Name du StackPanel correspondant) pour pouvoir y faire référence. Ensuite (le XAML est séquentiel, les déclarations doivent avoir lieu AVANT leur utilisation), on définit un Trigger qui portera sur la propriété Latitude de l'objet (principe : Latitude et Longitude sont nulles, ou non, en même temps). Ce Trigger est déclenché par la valeur spécifique "null" (référence nulle, balise spéciale <x:Null/> en XAML). L'action est de modifier (set) une propriété : un Setter sur l'objet nommé avant,

pour sa propriété "Visibility", à sa valeur "Hidden". Il faudrait théoriquement faire la même chose pour chaque Tag Exif intéressant (c'est juste un exemple).

8) Affichage de l'image en grand format
Ajoutez APRES la fermeture de la ListBox l'élément suivant :

```
            <Image Grid.Column="4" x:Name="image"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
                   Source="{Binding ElementName=listBox,
Path=SelectedItem.FullName}"/>
```

Ceci crée une image en grand format. Poser à "Stretch" ses alignements permet de l'afficher à la taille maximale possible (sans changer ses proportions), dans la colonne de contenu. Le Binding fait référence à la ListBox (qui a été nommée "listBox" précédemment... en théorie), spécifiquement à son Item SelectedItem (qui est null tant qu'aucun objet n'est sélectionné et prend la valeur du premier objet sélectionné quand il y en a au moins un) et accède à la propriété FullName dudit Item (qui est donc un JPGFileData) pour charger l'image.

V. Interface et Diaporama

1) Ajout d'un menu
Remontez un peu dans le XAML. On ajoute AVANT la Grid placée ligne 1 ceci :

```
        <Menu Grid.Row="0" IsMainMenu="True">
            <MenuItem Header="_Diaporama" Click="Diaporama_Click" />
            <MenuItem Header="_Edit" Click="EditItem_Click" />
        </Menu>
```

Cela crée un menu dans la première ligne (sur deux) de la grille principale de la fenêtre (celle qui fait 20 pixels de haut)... Ou du moins une "barre de menus", les items se comportant comme des boutons pour l'instant. Dans les MenuItem, le _ est à placer avant la lettre qui servira de raccourci clavier (pour déclencher l'item sans la souris, appuyer sur "tab" pour sélectionner le menu, puis sur la touche correspondante). "Click" fait référence à l'évémement Click, et la valeur doit être le nom d'une méthode gérant cet événement : on peut laisser l'IDE la créer à la main (et la renommer assez facilement ensuite), ou créer une méthode

```
    private void Diaporama_Click(object sender, RoutedEventArgs e)
```

dans MainWindow.xaml.cs a posteriori.

2) Fenêtre "Diaporama"
Créez une nouvelle fenêtre nommée Slideshow. Voici un début de code pour celle-ci :

```
  public partial class Slideshow : Window
    {
        public List<JPGFileData> Pictures { get; set; }
        public Slideshow(List<JPGFileData> pictures)
        {
            if (pictures == null) return;
            Pictures = pictures;
            InitializeComponent();
            WindowStyle = WindowStyle.None;
            WindowState = WindowState.Maximized;
            ResizeMode = ResizeMode.NoResize;
            KeyDown += Slideshow_KeyDown;
            MouseDown += Slideshow_MouseDown;
        }
        private void Slideshow_MouseDown(object sender,
MouseButtonEventArgs e)
```

```
        {
            throw new NotImplementedException();
        }

        private void Slideshow_KeyDown(object sender, KeyEventArgs e)
        {
            throw new NotImplementedException();
        }
    }
```

Côté interface, on se contentera d'une seule balise Image :

```
    <Image x:Name="myImage" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"/>
```

...à insérer dans la Grid de Slideshow.xaml. On peut ne rien changer d'autre, les propriétés sont modifiées par le code C#.

3) Affichage de la première image

Pour changer l'image affichée, on récupère la JPGFileData correspondante et on crée une URI permettant de charger une BitmapImage servant de source au contrôle Image créé précédement. Il suffit d'ajouter cette ligne à la fin du constructeur pour la première image :

```
            myImage.Source = new BitmapImage(new Uri(Pictures[0].FullName));
```

(myImage est le nom donné au contrôle image, qui devient accessible dans le code).
Pour lancer le diaporama, il suffit de revenir à la méthode gérant le clic sur l'item de menu correspondante, et d'ajouter cette ligne :

```
            new Slideshow((DataContext as FileSource).Files).ShowDialog();
```

Attention ! dans MainWindow.xaml.cs, la fenêtre dispose de la liste des fichiers en tant que DataContext, qui est générique et donc de type objet.
Il faut donc expliciter le type "maison" (FileSource) avant de pouvoir s'en servir.

Tester.

4) Timer pour le diaporama

Pour faire défiler les images, on commence par se donner l'indice de l'image courante dans un attribut entier :

```
    private int current = 0;
```

Ensuite, on extrait une méthode permettant d'avancer à l'image suivante :

```
        private void NextImage()
        {
            myImage.Source = new BitmapImage(new Uri(Pictures[current++].FullName));
            current %= Pictures.Count;
        }
```

(On peut utiliser cette méthode dans l'initialisation aussi).
Ensuite, on utilise un DispatcherTimer (par exemple) en attribut :

```
     private DispatcherTimer timer = new DispatcherTimer();
```

Et on ajoute dans le constructeur :

```
            timer.Interval = TimeSpan.FromSeconds(2);
            timer.Tick += Timer_Tick;
            timer.Start();
```

Avec comme gestion de l'événement "Tick" du timer (toutes les deux secondes ici) :

```
    private void Timer_Tick(object sender, EventArgs e)
        {
            NextImage();
        }
```

...C'est tout.

5) Contrôles clavier et souris
On peut s'ajouter de quoi gérer la souris et le clavier. Exemples : clic -> image suivante, espace -> pause ou reprise, Echap. -> quitter le diaporama.
Voilà le code :

```
        private void Slideshow_MouseDown(object sender,
MouseButtonEventArgs e)
        {
            NextImage();
        }

        private void Slideshow_KeyDown(object sender, KeyEventArgs e)
        {
            switch (e.Key)
            {
                case Key.Space:
                    if (timer.IsEnabled) timer.Stop();
                    else timer.Start();
                    break;
                case Key.Escape:
                    Close();
                    break;
            }
        }
```

Tester, vérifier.

VI. Contrôle "sélection de dossier"
1) Objectif
Le but est d'avoir dans la première colonne de contenu un (classique) sélecteur de fichiers "arborescent", permettant de "déplier" les répertoire avant de choisir le dossier images. Pour plus de réutilisabilité, ceci sera un contrôle utilisateur maison (adaptable à d'autres applications... en théorie).

2) Classe "répertoire"
Peu de choses à dire, remarquez qu'il s'agit d'un type structurellement inductif (un Dossier peut contenir des Dossiers) :

```
  public class DirectoryItem
    {
        public string Name { get; set; }
        public string FullName { get; set; }
        public List<DirectoryItem> Items { get; set; }
        public DirectoryItem()
        {
            Items = new List<DirectoryItem>();
        }
    }
```

3) Fournisseur de liste de répertoires
Méthode (de classe) à ajouter à la classe "FileSource" (...par exemple) :

```
    public static List<DirectoryItem> GetItems(string path)
        {
```

```
            var items = new List<DirectoryItem>();
            var dirInfo = new DirectoryInfo(path);
            foreach (var directory in dirInfo.GetDirectories())
            {
                var item = new DirectoryItem
                {
                    Name = directory.Name,
                    FullName = directory.FullName,
                    Items = GetItems(directory.FullName)
                };
                items.Add(item);
            }
            return items;
        }
```

4) Contrôle de navigation

Créer un nouveau contrôle utilisateur, "BrowserControl". Ajouter dans son interface ceci :

```
    <TreeView ItemsSource="{Binding}">
            <TreeView.ItemTemplate>
                <HierarchicalDataTemplate>
                    <TextBlock Text="{Binding Path=Name}"
ToolTip="{Binding Path=FullName}"/>
                </HierarchicalDataTemplate>
            </TreeView.ItemTemplate>
        </TreeView>
```

Cela crée une vue arborescente contenant une ligne par répertoire (avec le nom abrégé du répertoire affiché).

Visualiser le code source et ajouter cette ligne au constructeur :

```
      DataContext =
FileSource.GetItems(Environment.GetFolderPath(Environment.SpecialFolder.M
yPictures));
```

Cela initialisera la liste des répertoire au contenu de "Mes Images".

Pour ajouter le nouveau contrôle et l'insérer dans la colonne de gauche, plusieurs possibilités. Voici C# :

dans le constructeur de MainWindow.xaml.cs, ajouter :

```
      BrowserControl browserControl = new BrowserControl
            {
                HorizontalAlignment = HorizontalAlignment.Stretch,
                VerticalAlignment = VerticalAlignment.Stretch
            };
            contentGrid.Children.Add(browserControl);
            Grid.SetColumn(browserControl, 0);
```

(Ici, contentGrid devrait être le nom de la grille interne avec ses cinq colonnes.)

5) Gestion de l'événement "sélection d'un nouveau dossier"

Pour faire les tests, il est nécessaire d'avoir quelques dossiers dans "Mes Images"... si possible avec (différentes) images.

On souhaite gérer la sélection du dossier. Quelques étapes (plus ou moins) simples :

- Dans le BrowserControl, ajouter à la balise TreeView un gestionnaire pour SelectedItemChanged. On devrait avoir une méthode
correspondant dans le code C#.
- Ajouter un /événement/ au BrowserControl :

```
    public event RoutedEventHandler ItemChanged;
```
(Ceci ressemble à un attribut, mais non, c'est un "événement"... c'est différent.)
- Le code de la méthode :
```
    private void TreeView_SelectedItemChanged(object sender,
RoutedPropertyChangedEventArgs<object> e)
        {
            ItemChanged?.Invoke(sender, e);
        }
```
Explication : ItemChanged doit exister (être non nul) pour pouvoir être levé. Ceci correspond au déclenchement d'un
événement fait maison, qu'on peut ensuite intercepter (la fenêtre principale peut définir une réponse à cet événement
pour ce contrôle, maintenant).
- Interception de l'événement... Dans le code de MainWindow.xaml.cs, dans le constructeur, après avoir crée le BrowserControl, rajouter :
```
      browserControl.ItemChanged += BrowserControl_ItemChanged;
```
(avec la création de la méthode qui va avec).
- Gestion de l'événement : cette méthode est :
```
    private void BrowserControl_ItemChanged(object sender,
RoutedEventArgs e)
        {
            DataContext = new FileSource((((sender as
TreeView).SelectedItem) as DirectoryItem).FullName);
        }
```
(Deux coercitions de types sont nécessaires pour récupérer les valeurs spécifiques attendues ; il est aussi possible
de faire une récupération de l'argument e.)

6) Ajoutez ce que vous souhaitez !